

conventions, and architecturally unaddressable when the processor pipeline is executing in the other architecture or convention.

133. (new) The computer processor of claim 113, further comprising circuitry designed to raise an exception when execution flows or transfers from a region whose indicator element indicates one architecture or execution convention to another.

REMARKS

This paper responds to the Office Action of February 20, 2002.

Filed concurrently herewith is a Petition for Extension of Time for one month, which extends the time to respond through June 20, 2002. Accordingly, this response is timely filed. In the event that any further extension of time is required, Applicant petitions for that extension of time required to make this response timely.

Applicant respectfully requests reconsideration of the application. Claims 1-133 are now pending, a total of 133 claims. Claims 1, 4, 22, 37, 51, 61, 87, 94, 96, 104 and 113 are independent. Claims 1-4, 14, 15, 18, 20-24, 27, 33, 34, 37, 38, 42, 44, 47, 50, 53-57, 59, 61-69, 73, 76, 77, 84, 85, 87, 92 and 94 are amended. New claims 96-133 are added to better claim the invention. Claims 1-95 are rejected under § 102 and § 103, over U.S. Pat. No. 5,481,684 to Richter.

Applicant thanks the Examiner for his thorough consideration of a large application.

I. Formal Drawings

The Notice of Draftsperson's Patent Drawing Review indicates that the Review is based on the drawings filed "8/30/99." Formal drawings were filed November 14, 2000. Applicant requests approval of these drawings.

II. Title

Paragraph 2 of the Office Action requests an amendment to the title. Applicant appreciates the Examiner's observation that the title was excessively narrow with respect to several of the claims. The title has been amended to more accurately embrace all of the subject matter recited in the claims.

III. Amendments to the claims

The amendment to claim 21 responds to a rejection under § 112 ¶ 2. The amendment does not narrow the scope of the claims.

Most amendments to the claims correct minor informalities, and are neither narrowing nor made pursuant to any statutory requirement raised by the Office Action.

IV. Claims 4 and 37

Paragraph 7 of the Action rejects claim 4 over Richter '684. As now amended, claim 4 recites as follows:

4. A method, comprising the steps of:

· executing instructions fetched from first and second regions of a memory of a computer, the instructions of the first and second regions being coded for execution by computers of first and second architectures or following first and second data storage conventions, respectively, the memory regions having associated first and second indicator elements, the indicator elements each having a value indicating the architecture or data storage convention under which instructions from the associated region are to be executed, the indicator elements being architecturally addressable when the processor pipeline is executing under one of the architectures or data storage conventions, and architecturally unaddressable when the computer is executing under the other architecture or data storage convention;

when execution of the instruction data flows or transfers from the first region to the second, adapting the computer for execution in the second architecture or convention.

The Action asserts that Fig. 4 and col. 8, lines 61 to col. 9, line 9 of Richter '684 correspond to the "indicator elements" of the claim, apparently referring to the "segment descriptors" of Richter '684.

Richter '684 makes clear that the segment descriptors are architecturally addressable in both his CISC and RISC modes. The discussion of access of these descriptors from CISC mode is discussed at length, from col. 6, line 40 to col. 10, line 45. To take one of the most extreme examples, at col. 10, line 9-11, Richter '684 suggests that the CISC architecture has been "extended" to have accessibility to even those segment descriptors "that can hold RISC code rather than just CISC code." Conversely, at col. 10, lines 63-67, Richter '684 describes that his RISC architecture is extended with "extended instructions" that "offer access to all of the system resources," which necessarily includes the segment descriptors.

Thus, there is no “architecture or data storage convention” in which the segment descriptors are “architecturally unaddressable,” as required by claim 4.

For this reason, claim 4 is distinguished from Richter '684, and is patentable over that reference. Claim 37 recites similar language, and is patentable for similar reasons.

Claims 5-12 and 14-21 are allowable for the same reasons discussed above with respect to independent claims 4 and 37, from which they depend. These claims also recite additional patentable features.

V. Claim 22

The limitations of claim 22 appear not to be addressed in the Office Action. Claim 22 is now rewritten into independent form by including many of the limitations of its parent claims. However, some limitations have been removed, and claim 22 is now somewhat broader. As now amended, claim 22 recites:

22. A method, comprising the steps of:
executing instructions fetched from first and second regions of a memory of a computer, the instructions of the first and second regions being coded for execution by computers following first and second data storage conventions, the memory regions having associated first and second indicator elements, the indicator elements each having a value indicating the data storage convention under which instructions from the associated region are to be executed;
recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

Claim 22 recites that a computer is able to recognize “when program execution has flowed or transferred from a region” using one storage convention, e.g., a calling convention or “endianness,” to a region using another convention, and “in response to the recognition” to “[alter] the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to [the] program context under the first data storage convention.” For example, this claim might be met by some mechanism for recognizing when a program flows from a region of code that uses “big endian” assumptions (embodied in shift instructions, addressing the same data in memory using loads or stores of differing sizes,

etc.) to a “little endian” region (col. 9, line 18-26); or if program flowed from a region that uses the calling convention enforced by a compiler for the Intel architecture (where most function call parameters passed on the stack) to a region that uses the calling convention enforced by a compiler for the IBM PowerPC architecture (where most function call parameters are passed in registers), and then took some action to readjust the layout of the data in memory from the layout assumed by the transferor to the layout assumed by the transferee.

But Richter '684 teaches no such thing. Indeed, Richter '684 recognizes that cross-architecture calls are a difficult problem. The only solution for the problem that he proposes is to provide two different entry points for routines that can be called by either RISC or CISC code. (col. 6, lines 1-14). Richter '684 teaches nothing analogous to “altering the data storage content of the computer” in the manner recited in claim 22.

For this reason, claim 22 is patentable over Richter '684. Claims 2, 18, 42, 54, 73, 76, 77 and 85 recite limitations that are analogous, and they are patentable for analogous reasons.

Claims 19-21, 23-36, 43-48, 55-60, 74, 75 and 78-84 are dependent on these claims, and allowable therewith. These claims also recite additional patentable features.

VI. Claims 51 and 61

Claim 51 is rejected over Richter '684 at paragraph 9 of the Action. Claim 51 is unamended, and recites as follows:

51. A method, comprising:

storing instructions in pages of a computer memory managed by a virtual memory manager, the instruction data of the pages being coded for execution by, respectively, computers of two different architectures and/or under two different execution conventions;

in association with pages of the memory, storing corresponding indicator elements indicating the architecture or convention in which the instructions of the pages are to be executed;

executing instructions from the pages in a common processor, the processor designed, responsive to the page indicator elements, to execute instructions in the architecture or under the convention indicated by the indicator element corresponding to the instruction's page.

As discussed in section X, above, the closest analog to the “indicator elements” of claim 13 appears to be the “segment descriptors” and “segment registers” discussed in Richter '684 at col. 8, line 61 to col. 9 line 26. However, Richter's “segment descriptors” and “segment registers”

etc.) to a “little endian” region (col. 9, line 18-26); or if program flowed from a region that uses the calling convention enforced by a compiler for the Intel architecture (where most function call parameters passed on the stack) to a region that uses the calling convention enforced by a compiler for the IBM PowerPC architecture (where most function call parameters are passed in registers), and then took some action to readjust the layout of the data in memory from the layout assumed by the transferor to the layout assumed by the transferee.

But Richter '684 teaches no such thing. Indeed, Richter '684 recognizes that cross-architecture calls are a difficult problem. The only solution for the problem that he proposes is to provide two different entry points for routines that can be called by either RISC or CISC code. (col. 6, lines 1-14). Richter '684 teaches nothing analogous to “altering the data storage content of the computer” in the manner recited in claim 22.

For this reason, claim 22 is patentable over Richter '684. Claims 2, 18, 42, 54, 73, 76, 77 and 85 recite limitations that are analogous, and they are patentable for analogous reasons.

Claims 19-21, 23-36, 43-48, 55-60, 74, 75 and 78-84 are dependent on these claims, and allowable therewith. These claims also recite additional patentable features.

VI. Claims 51 and 61

Claim 51 is rejected over Richter '684 at paragraph 9 of the Action. Claim 51 is unamended, and recites as follows:

51. A method, comprising:

storing instructions in pages of a computer memory managed by a virtual memory manager, the instruction data of the pages being coded for execution by, respectively, computers of two different architectures and/or under two different execution conventions;

in association with pages of the memory, storing corresponding indicator elements indicating the architecture or convention in which the instructions of the pages are to be executed;

executing instructions from the pages in a common processor, the processor designed, responsive to the page indicator elements, to execute instructions in the architecture or under the convention indicated by the indicator element corresponding to the instruction's page.

The structure in Richter '684 most nearly-analogous to the “indicator elements” of claim 51 appears to be the “segment descriptors” and “segment registers” discussed in Richter '684 at col. 8, line 61 to col. 9 line 26. However, Richter's “segment descriptors” and “segment registers”

are associated with segments. See, *e.g.*, Richter '684 at col. 6, line 40 to col. 8, line 25. In contrast, claim 51 recites that the indicator elements are associated with "pages."

Because claim 51 is distinguished from Richter '684, claim 51 is patentable over that reference.

Claims 1, 5, 11, 19, 30, 61, and 89 recite analogous language and are patentable for analogous reasons.

Claims 2, 3, 6-9, 31, 32, 52-60, 62-86, 90 and 91 are dependent on these claims, and allowable therewith. These claims also recite further patentable features.

VII. Claim 87

Paragraphs 16 and 17 of the Office Action assert that claim 87 is obvious over Richter '684. Claim 87 reads as follows:

87. A method, comprising the steps of:

executing a control-transfer instruction under a first execution context of a computer, the instruction being architecturally defined to transfer control directly to a destination instruction for execution in a second execution context of the computer;

before executing the destination instruction, altering the data storage content of the computer to establish a program context under the second execution context that is logically equivalent to the context of the computer as interpreted under the first execution context, the reconfiguring including at least one data movement operation not included in the architectural definition of the control-transfer instruction.

The Office Action characterizes Richter '684 as indicating "the necessity of transforming aspects specific to one architecture into the appropriate aspect specific to the other architecture."¹ Even if, for sake of argument, this characterization were correct, it cannot be disputed that Richter '684 fails to teach any technique for actually performing that "transforming."² Richter '684 only demonstrates the difficulty that the art has faced with this problem. For example, at col. 6, lines 4-14, Richter '684 discusses the need to provide two different entry points for every service routine in an operating system, and how undesirable that solution is. At col. 6, lines 24-

¹ Applicant disagrees with this characterization, but in view of the alternative argument presented in the body of this paper, this argument will be saved for another day.

² The Action makes no showing of "reasonable expectation of success" as required by MPEP § 2143.02. Without that showing, any obviousness rejection is incomplete.

26, Richter '684 states that "ideally" there would be only one entry point for each routine – but he teaches no technique for achieving this "ideal." As another example, Richter '684 never discusses any solution to the mismatch between the memory-stack-based calling convention conventionally used in his X86 CISC mode and the register-based calling convention conventionally used in the PowerPC RISC mode.

The one portion of a solution discussed by Richter '684, the switch from big-endian to little-endian, col. 9, lines 17-26, does not involve "at least one data movement operation" as recited in claim 87.

MPEP § 2143.03 instructs that a claim can never be obvious when it recites a limitation absent from the art. Because the claim does recite such a limitation, claim 87 is not obvious.

Claims 88-93 are dependent on claim 87, and allowable therewith. These claims also recite further patentable features.

VIII. Claims 94 and 96

Paragraph 15 of the Office Action asserts that claim 94 is met by Richter at col. 13 line 48 to col. 14, line 5. Claim 94 recites as follows:

94. A method, comprising the steps of:

executing a section of computer object code twice, without modification of the code section between the two executions, the code section materializing a destination address into a register and being architecturally defined to directly transfer control indirectly through the register to the destination address, the two executions materializing two different destination addresses;

the two destination code sections at the two materialized destination addresses being coded in two distinct instruction sets, neither instruction set being a subset of the other.

Claim 94 is directed to a computer that seamlessly handles the transition between two instruction sets, at least well enough to meet this test – a given point in the program can transfer to another point in the program without care for the instruction set that the transferee is coded in. Claim 96 recites a similar invention, a computer that executes two different instruction sets, each with an associated data storage convention.

Richter '684, at col. 13 line 48 to col. 14 line 5, teaches only that the hardware has some capability to execute two different instruction sets. This section teaches nothing to indicate that a transferor in one instruction set can simply transfer control to a transferee, oblivious of what the transferee routine might be, without carefully preparing to transfer to the transferee in the correct

way, with the data in the locations that the transferee expects, etc. There are at least three ways to do this: (a) rewrite the compilers for either the CISC and RISC architecture (or both) so that they use a common data storage convention that allows free transfer between, (b) manually tailor each crossing point between CISC code and RISC code, or (c) provide an automatic readjustment capability that acts at the transfer boundary, in each case to make sure that the data expectations of the transferor corresponds to the data expectations of all transferees. Richter '684 does not even suggest doing any of the three, let alone teach a method for doing any of them. Without that teaching, a computer cannot perform as recited in claim 94.

Claim 96 is added by amendment in this paper, as an apparatus companion of claim 94. Claim 96 is patentable for analogous reasons.

Claims 95 and 97-105 are allowable with claims 94 and 96, and recite further patentable features.

IX. Claims 104 and 113

Claims 104-133 are added in this amendment to better protect the invention. They have not been rejected. Claim 104 may be considered representative:

104. A method, comprising the steps of:

executing instructions fetched from first and second regions of a single address space of the memory of a computer, the instructions of the first and second regions being coded for execution by computers of first and second architectures or following first and second data storage conventions, respectively, the memory regions having associated first and second modifiable indicator elements, a hardware structure for storing the indicator elements enforcing a requirement that the memory regions be necessarily disjoint, the modifiable indicator elements each having a value indicating the architecture or data storage convention under which instructions from the associated region are to be executed;

when execution of the instruction data flows or transfers from the first region to the second, adapting the computer for execution in the second architecture or convention.

The claim recites that the computer hardware requires that "the memory regions be necessarily disjoint." In contrast, the "indicator elements" of Richter '684 correspond to segments of the Intel x86 architecture (col. 10, lines 8-18). As is well known in the art, Intel segments may – and frequently do – overlap. For example, it is fairly common for Intel programs to run with the code, data and stack segments exactly overlaying each other.

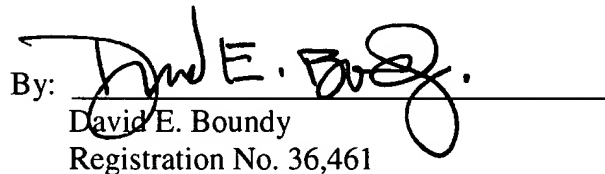
For this reason, claims 104 and 113 are patentable over Richter '684. Claims 105-112 and 114-133 are allowable therewith; further, these dependent claims recite further patentable limitations.

In view of the amendments and remarks, Applicant respectfully submits that the claims are in condition for allowance. Applicant requests that the application be passed to issue in due course. The Examiner is urged to telephone Applicant's undersigned counsel at the number noted below if it will advance the prosecution of this application, or with any suggestion to resolve any condition that would impede allowance. Enclosed is Petition for Extension of Time for one month. In the event that any further extension of time is required, Applicant petitions for that extension of time required to make this response timely. Kindly charge any additional fee, or credit any surplus, to Deposit Account 50-0675, Order No. 5231.03-4000.

Respectfully submitted,

SCHULTE ROTH & ZABEL

Dated: June 20, 2002

By: 
David E. Boundy
Registration No. 36,461

Mailing Address:
SCHULTE ROTH & ZABEL
919 Third Avenue
New York, New York 10022
(212) 756-2000
(212) 593-5955 Telecopier

VERSION OF REWRITTEN CLAIMS MARKED UP TO SHOW CHANGES

1. (amended) A computer, comprising:

a processor pipeline designed to alternately execute instructions coded for first and second different computer architectures or coded to implement first and second different processing conventions;

a memory for storing instructions for execution by the processor pipeline, the memory being divided into pages for management by a virtual memory manager, a single address space of the memory having first and second pages;

a memory unit designed to fetch instructions from the memory for execution by the pipeline, and to fetch stored indicator elements associated with respective memory pages of the single address space from which the instructions are to be fetched, each indicator element designed to store an indication of which of two different computer architectures and/or execution conventions under which instruction data of the associated page are to be executed by the processor pipeline, the indicator elements being architecturally addressable when the processor pipeline is executing under one of the architectures or data storage conventions, and architecturally unaddressable when the computer is executing under the other architecture or data storage convention;

the memory unit and/or processor pipeline further designed to recognize an execution flow from the first page, whose associated indicator element indicates the first architecture or execution convention, to the second page, whose associated indicator element indicates the first architecture or execution convention, and in response to the recognizing, to adapt a processing mode of the processor pipeline or a storage content of the memory to effect execution of instructions in the architecture and/or under the convention indicated by the indicator element corresponding to the instruction's page.

2. (amended) The computer [method] of claim 1:

wherein the two architectures are two instruction set architectures;

and wherein the adapting [step] includes controlling instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the indicator elements.

3. (amended) The computer [method] of claim 1, wherein the two conventions are first and second calling conventions, and further comprising:

hardware and/or software designed to recognize [recognizing] when program execution has flowed or transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, to alter [altering] the data storage content of the computer to create a program context under the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention.

4. (amended) A method, comprising the steps of:

executing instructions fetched from first and second regions of a [single address space of the] memory of a computer, the instructions of the first and second regions being coded for execution by computers of first and second architectures or following first and second data storage conventions, respectively, the memory regions having associated first and second indicator elements, the indicator elements each having a value indicating the architecture or data storage convention under which instructions from the associated region are to be executed, the indicator elements being architecturally addressable when the processor pipeline is executing under one of the architectures or data storage conventions, and architecturally unaddressable when the computer is executing under the other architecture or data storage convention;

when execution of the instruction data flows or transfers from the first region to the second, adapting the computer for execution in the second architecture or convention.

14. (amended) The method of claim 10, wherein a mode of execution of the instructions is changed without software intervention when execution flows or transfers from the first region to the second.

15. (amended) The method of claim 10, wherein execution of the computer takes an exception when execution flows or transfers from the first region to the second.

18. (amended) The method of claim 10, wherein the two conventions are first and second data storage conventions, and further comprising the step of:

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

20. (amended) The method of claim 18, further comprising the steps of:
classifying control-flow instructions of a computer instruction set into a plurality of classes; and
during execution of a program on a computer, as part of the execution of instructions of the instruction set, updating a record of the class of the classified control-flow instruction most recently executed;
the adjusting process being determined, at least in part, by the instruction class record.

21. (amended) The method of claim 18, wherein:
the instruction data coded for execution by a first of the two instruction set architectures observes a data storage convention associated with the first architecture, and instruction data coded for execution by a second of the two instruction set architectures observes a second, different, data storage convention associated with the second architecture, a single indicator element indicating both the instruction set architecture and the data storage convention;
and further comprising the step of [,] recognizing when program execution flows or transfers from a region using the first instruction set architecture to a region using the second instruction set architecture, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second.[.]

22. (amended) A [The] method [of claim 4, wherein the two conventions are first and second data storage conventions, and further], comprising the steps of:
executing instructions fetched from first and second regions of a memory of a computer,
the instructions of the first and second regions being coded for execution by computers following
first and second data storage conventions, the memory regions having associated first and second

indicator elements, the indicator elements each having a value indicating the data storage convention under which instructions from the associated region are to be executed;

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

23. (amended) The method of claim 22, further comprising the step of:

overlaying the logical resources of the first and second instruction set architectures onto the physical resources of the computer according to a mapping that assigns corresponding resources of the two architectures to a common physical resource of a computer when the resources serve analogous functions in the calling conventions of the two architectures.

24. (amended) The method of claim 22, wherein the adjusting step further comprises:

altering a bit representation of a datum from a first representation in the first convention to a second representation in the second convention, the alteration of representation being chosen to preserve the meaning of the datum across the change in data storage [execution] convention.

27. (amended) The method of claim 22, wherein

a rule for copying data from the first location to the second is determined by examining a descriptor associated with the location of execution before the recognized execution flow or transfer.

33. (amended) The method of claim 28, further comprising the step of:

taking a processor exception in response to the recognition, a handler for the exception programmed to copy a datum from a first location to a second location, the first location having a use under the first data storage convention analogous to the use of the second location under the second data storage convention.

34. (amended) The method of claim 22, further comprising the step of:
classifying control-flow instructions of a computer instruction set into a plurality of
classes; and
during execution of a program on a computer, as part of the execution of instructions of
the instruction set, updating a record of the class of the classified control-flow instruction most
recently executed;
the adjusting process being determined, at least in part, by the instruction class record.

37. (amended) A computer processor, comprising:
a processor pipeline configured to alternately execute instructions of computers of two
different architectures or processing conventions; and
a memory unit designed to fetch instructions from a computer memory for execution by
the pipeline, and to fetch stored indicator elements associated with respective memory regions of
a single address space from which the instructions are to be fetched, each indicator element
designed to store an indication of the architecture or execution convention under which the
instruction data of the associated region are to be executed by the processor pipeline, the
indicator elements being maintained in storage that is architecturally addressable when the
processor pipeline is executing in one of the computer architectures or processing conventions,
and architecturally unaddressable when the processor pipeline is executing in the other
architecture or convention;
the memory unit and/or processor pipeline further designed to recognize an execution
flow or transfer from a region whose indicator element indicates one architecture or execution
convention to another.

38. (amended) The computer processor [method] of claim 37, wherein the indicator
elements are stored in a table of indicator elements distinct from a primary address translation
table used by the virtual memory manager, the indicator elements of the table associated with
corresponding pages of the memory.

42. (amended) The computer processor of claim 40:

each indicator element being further designed to store an indication of a calling convention under which the instruction data of the associated region are coded for execution by the processor pipeline;

and further comprising software programmed to alter the data storage content of a computer using the computer processor to create a program context under the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention;

the memory unit further designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, to invoke the transition management software.

44. (amended) The computer processor of claim 42, wherein the memory unit and software are designed to effect a transition between instruction boundaries, between execution in a region coded in the first instruction set using the first calling convention to execution in a region coded in the second instruction set using the second calling convention, so that code at the source of the flow or transfer may effect the execution transition without being specially coded for code at the destination.

47. (amended) The computer processor of claim 42, wherein
a rule for altering the data storage content from the first calling convention to the second is determined by examining a descriptor associated with the location of execution before the recognized execution flow or transfer.

50. (amended) The computer processor of claim 40, further comprising circuitry designed to raise an exception when the computer recognizes that [the] execution has flowed or transferred from a region whose indicator element indicates one architecture or execution convention to another [is recognized].

53. (amended) The method of claim 51, wherein the two architectures are two instruction set architectures, and further comprising the step of:

controlling the instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the indicator element.

54. (amended) The method of claim 51, wherein the two conventions are first and second data storage conventions, and further comprising the step of:

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

55. (amended) The method of claim 54, further comprising the steps of:

storing instruction data in a third page, the instruction data of the third page being coded for execution by one of the two architectures, and observing a data storage convention associated with the other of the two architectures;

storing indicator elements indicating the data storage convention observed by the instructions of the respective pages; and

recognizing each transition of program execution from a page using the first data storage convention to a page using the second data storage convention, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second, and vice-versa.

56. (amended) The method of claim 53, wherein:

the instruction data coded for execution by a first of the two instruction set architectures observes a data storage convention associated with the first architecture, and instruction data coded for execution by a second of the two instruction set architectures observes a second, different, data storage convention associated with the second architecture, a single indicator element indicating both the instruction set architecture and the data storage convention of the associated page;

and further comprising the step of [,] recognizing when program execution flows or transfers from a page using the first instruction set architecture to a page using the second instruction set architecture, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second.

57. (amended) The method of claim 54, wherein the two conventions are a register-based calling convention and a memory stack-based calling convention, and further comprising the step of:

recognizing when program execution has flowed or transferred from a page using the register-based calling convention to a page using the memory stack-based convention, and in response to the recognition, adjusting the data storage content of the computer from the first calling convention to the second.

59. (amended) The method of claim 54, wherein
a rule for copying data from the first location to the second is determined by examining a descriptor associated with the location of execution before the recognized execution flow or transfer.

61. (amended) A microprocessor chip, comprising:
an instruction unit, configured to fetch instructions from a memory managed by the virtual memory manager, and configured to execute instructions coded for first and second different computer architectures or coded to implement first and second different data storage conventions;
the microprocessor chip being designed (a) to retrieve indicator elements stored in association with respective pages of the memory, each indicator element indicating the architecture or convention in which the instructions of the page are to be executed, and (b) to recognize when instruction execution has flowed or transferred from a page of the first architecture or convention to a page of the second, as indicted by the respective associated indicator elements, and (c) to alter a processing mode of the instruction unit or a storage content of the memory to effect execution of instructions in accord with the indicator element associated with the page of the second architecture or convention.

62. (amended) The microprocessor chip [method] of claim 61, wherein the indicator elements are stored in virtual address translation table entries.

63. (amended) The microprocessor chip [method] of claim 61, wherein the indicator elements are stored in a table distinct from a primary address translation table used by a virtual memory manager, the indicator elements of the table being stored in association with respective pages of the memory.

64. (amended) The microprocessor chip [method] of claim 61, wherein the indicator elements are stored in association with respective physical page frames.

65. (amended) The microprocessor chip [method] of claim 61, wherein the indicator elements are stored in association with respective virtual pages.

66. (amended) The microprocessor chip [method] of claim 61, wherein the indicator elements are stored in entries of a translation look-aside buffer.

67. (amended) The microprocessor chip [method] of claim 61, wherein the indicator elements are stored in an instruction cache.

68. (amended) The microprocessor chip of claim 61, wherein a mode of execution of the instructions is changed without software intervention when execution flows or transfers from the first region to the second.

69. (amended) The microprocessor chip of claim 61, the microprocessor chip being designed to raise an exception when execution flows or transfers from the first region to the second;

and further comprising exception handler software programmed to handle the exception by explicitly controlling a mode of execution of the instructions.

73 (amended) The microprocessor chip of claim 71:

each indicator element being further designed to store an indication of a data storage convention under which the instruction data of the associated page are coded for execution by the instruction unit;

and further comprising software programmed to alter the data storage content of a computer using the microprocessor chip, to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention;

the microprocessor chip further designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, to invoke the transition management software.

76. (amended) The microprocessor chip of claim 71, further comprising hardware and/or software designed:

(a) to retrieve calling convention indicator elements stored in association with respective pages of the memory, each calling convention indicator element indicating which of a register-based calling convention or a memory stack-based calling convention is observed by instructions of the page;

(b) to recognize when instruction execution has flowed or transferred from a page of a memory-based convention to a page of the register-based calling convention, as indicated by the calling convention indicator elements associated with the respective pages, and

(c) in response to the recognition, to alter a storage content of the computer to create a program context under the register-based convention logically equivalent to a pre-alteration program context under the memory-based convention.

77. (amended) The microprocessor chip of claim 61:

wherein the two conventions are first and second data storage conventions;

and further comprising software programmed to alter the data storage content of a computer using the microprocessor chip, to create a program context under the second data

storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention;

the microprocessor chip being further designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, to invoke the transition management software.

84. (amended) The microprocessor chip of claim 79, further comprising:

software and/or hardware designed to copy a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, the software and/or hardware for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program.[.]

85. (amended) The microprocessor chip of claim 61, further comprising hardware and/or software designed:

(a) to retrieve calling convention indicator elements stored in association with respective pages of the memory, each calling convention indicator element indicating which of a register-based calling convention or a memory stack-based calling convention is observed by instructions of the page;

(b) to recognize when instruction execution has flowed or transferred from a page using the register-based calling convention to a page using the memory stack-based convention, as indicated by the calling convention indicator elements associated with the respective pages, and

(c) in response to the recognition, to alter a storage content of the computer to create a program context under the memory-based convention logically equivalent to a pre-alteration program context under the register-based convention.

87. (amended) A method, comprising the steps of:

executing a control-transfer instruction under a first execution context of a computer, the instruction being architecturally defined to transfer control directly to a destination instruction for execution in a second execution context of the computer;

before executing the destination instruction, altering the data storage content of the computer to establish a program context under the second execution context that is logically equivalent to the context of the computer as interpreted under the first execution context, the reconfiguring including at least one data movement operation not included in the architectural definition of the control-transfer instruction.

[copying data from a general register to a memory stack; and]

[copying data from a memory stack to a general register.]

92. (D25) The method of claim 87, further comprising the step of:

altering a bit representation of a datum from a first representation in the first context [convention] to a second representation in the second context [convention], the alteration of representation being chosen to preserve the meaning of the datum across the change in execution context [convention].

94. (amended) A method, comprising the steps of:

executing a section of computer object code twice, without modification of the code section between the two executions, the code section materializing a destination address into a register and being architecturally defined to directly transfer control indirectly through the register to the destination address, the two executions materializing two different destination addresses;

the two destination code sections at the two materialized destination addresses being coded in two distinct instruction sets, neither instruction set being a subset of the other.

Kindly add the following new claims.

96. (new) A microprocessor chip, comprising:

two instruction decoders designed to decode instructions of first and second instruction sets, respectively, and circuitry of a single instruction pipeline designed to execute the instructions decoded by either of the two instruction decoders;

circuitry and/or software designed to detect when execution flows or transfers control from code coded in one instruction set to code coded in the other, program code in the first and second instruction sets using first and second different data storage conventions, respectively; and

circuitry and/or software designed to respond to the detection by altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

97. (new) The computer processor of claim 96, wherein the memory unit and software are designed to effect a transition between instruction boundaries, between execution in a region coded in the first instruction set using the first calling convention to execution in a region coded in the second instruction set using the second calling convention, so that code at the source of the flow or transfer may effect the execution transition without being specially coded for code at the destination.

98. (new) The microprocessor chip of claim 96, wherein the two conventions are two calling conventions.

99. (new) The computer processor of claim 98, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

100. (new) The microprocessor chip of claim 96, further comprising:

software and/or hardware designed to copy a datum from a first location to a second location, the first location having a use under the first calling convention analogous to the use of the second location under the second calling convention.

101. (new) The microprocessor chip of claim 100, further comprising:
software and/or hardware designed to copy a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, the software and/or hardware for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program.

102. (new) The computer processor of claim 96, wherein
a rule for altering the data storage content from the first calling convention to the second is determined based on an instruction at the location of execution at the source of the recognized execution flow or transfer.

103. (new) The computer processor of claim 96, wherein
a rule for altering the data storage content from the first calling convention to the second is determined by examining a descriptor associated with the location of execution before the recognized execution flow or transfer.

104. (new) A method, comprising the steps of:
executing instructions fetched from first and second regions of a single address space of the memory of a computer, the instructions of the first and second regions being coded for execution by computers of first and second architectures or following first and second data storage conventions, respectively, the memory regions having associated first and second modifiable indicator elements, a hardware structure for storing the indicator elements enforcing a requirement that the memory regions be necessarily disjoint, the modifiable indicator elements each having a value indicating the architecture or data storage convention under which instructions from the associated region are to be executed;
when execution of the instruction data flows or transfers from the first region to the second, adapting the computer for execution in the second architecture or convention.

105. (new) The method of claim 104, wherein:
the regions are pages managed by a virtual memory manager.

106. (new) The method of claim 105, wherein the modifiable indicator elements are
stored in a table, each modifiable indicator element associated with a corresponding physical
page frame.

107. (new) The method of claim 105, wherein the entries are entries of a translation
look-aside buffer.

108. (new) The method of claim 104:
wherein the two architectures are two instruction set architectures;
and wherein the adapting step includes controlling instruction execution hardware of the
computer to interpret the instructions according to the two instruction set architectures according
to the modifiable indicator elements.

109. (new) The method of claim 108, wherein:
one of the regions stores an off-the-shelf operating system binary coded in an instruction
set non-native to the computer, the non-native instruction set providing access to a reduced
subset of the resources of the computer.

110. (new) The method of claim 108, wherein the two conventions are first and second
data storage conventions, and further comprising the step of:
recognizing when program execution has flowed or transferred from a region whose
modifiable indicator element indicates the first data storage convention to a region whose
modifiable indicator element indicates the second data storage convention, and in response to the
recognition, altering the data storage content of the computer to create a program context under
the second data storage convention that is logically equivalent to a pre-alteration program
context under the first data storage convention.

111. (new) The method of claim 104, wherein the two conventions are first and second data storage conventions, and further comprising the step of:

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

112. (new) The method of claim 111, wherein:

one of the two data storage conventions is a register-based calling convention, and the other data storage convention is a memory stack-based calling convention.

113. (new) A computer processor, comprising:

a processor pipeline configured to alternately execute instructions of computers of two different architectures or processing conventions; and

a memory unit designed to fetch instructions from a computer memory for execution by the pipeline, and to fetch stored indicator elements associated with respective necessarily-disjoint memory regions of a single address space from which the instructions are to be fetched, each indicator element designed to store an indication of the architecture or execution convention under which the instruction data of the associated region are to be executed by the processor pipeline;

the memory unit and/or processor pipeline further designed to recognize an execution flow or transfer from a region whose indicator element indicates one architecture or execution convention to another.

114. (new) The computer processor of claim 113, wherein the two architectures are two instruction set architectures, and further comprising:

processor pipeline control circuitry designed to control the processor pipeline to effect interpretation of the instructions under the two instruction set architectures alternately, according to the associated indicator elements.

115. (new) The computer processor of claim 114, further comprising:
software programmed to manage a transition between the execution of a program executing in the first instruction set architecture, being an instruction set architecture native to the computer processor, and execution of an off-the-shelf operating system coded in the second instruction set, being an instruction set non-native to the computer, providing access to a reduced subset of the resources of the computer.

116. (new) The computer processor of claim 114:
each indicator element being further designed to store an indication of a calling convention under which the instruction data of the associated region are coded for execution by the processor pipeline;

and further comprising software programmed to alter the data storage content of a computer using the computer processor to create a program context under the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention;

the memory unit further designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, to invoke the transition management software.

117. (new) The computer processor of claim 116, wherein the memory unit is designed to recognize a single indicator element to indicate both the instruction set architecture and calling convention of a region.

118. (new) The computer processor of claim 113, wherein the two conventions are first and second calling conventions, and further comprising:

hardware and/or software designed to recognize when program execution has flowed or transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, to alter the data storage content of the computer to create a program context under

the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention.

119. (new) The computer processor of claim 118, wherein the memory unit and software are designed to effect a transition between instruction boundaries, between execution in a region coded in the first instruction set using the first calling convention to execution in a region coded in the second instruction set using the second calling convention, so that code at the source of the flow or transfer may effect the execution transition without being specially coded for code at the destination.

120. (new) The computer processor of claim 118, wherein the two conventions are two calling conventions.

121. (new) The computer processor of claim 118, wherein:
one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

122. (new) The computer processor of claim 118, wherein the logical resources for support of the first and second instruction set architectures are overlaid on the physical resources of the computer processor according to a mapping that assigns corresponding resources of the two architectures to a common physical resource when the resources serve analogous functions in the calling conventions of the two architectures.

123. (new) The computer processor of claim 118, further comprising:
a transition manager designed to effect a transition between execution under the first calling convention to execution under the second calling convention, the transition manager designed to alter a bit representation of a datum from a first representation to a second representation, the alteration of representation being chosen to preserve the meaning of the datum across the change in calling convention.

124. (new) The computer processor of claim 118, further comprising:
software and/or hardware designed to copy a datum from a first location to a second location, the first location having a use under the first calling convention analogous to the use of the second location under the second calling convention.

125. (new) The computer processor of claim 124, further comprising:
software and/or hardware designed to copy a datum from a third location to a fourth, the third location having a use under the first calling convention analogous to the use of the third location under the first calling convention and to the fourth location under the second calling convention, the software and/or hardware for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program.

126. (new) The computer processor of claim 118, wherein
a rule for altering the data storage content from the first calling convention to the second is determined by examining a descriptor associated with the location of execution before the recognized execution flow or transfer.

127. (new) The computer processor of claim 118, wherein control-flow instructions of the instruction set are classified into a plurality of classes; and
the processor pipeline updates a record of the class of the classified control-flow instruction most recently executed;
the storage alteration process being determined, at least in part, by the instruction class record.

128. (new) The computer processor of claim 113, wherein:
the regions are pages managed by a virtual memory manager.

129. (new) The computer processor of claim 113, wherein the indicator elements are stored in virtual address translation table entries.

130. (new) The computer processor of claim 113, wherein the indicator elements are stored in a table of indicator elements distinct from a primary address translation table used by

the virtual memory manager, the indicator elements of the table associated with corresponding pages of the memory.

131. (new) The computer processor of claim 113, wherein the indicator elements are stored in respective entries of a table whose entries are associated with corresponding physical page frames.

132. (new) The computer processor of claim 113, wherein:
the indicator elements are stored in storage that is architecturally addressable when the processor pipeline is executing in one of the computer architectures or processing conventions, and architecturally unaddressable when the processor pipeline is executing in the other architecture or convention.

133. (new) The computer processor of claim 113, further comprising circuitry designed to raise an exception when execution flows or transfers from a region whose indicator element indicates one architecture or execution convention to another.